SEPARATE READ AND WRITE SERVERS IN A DISTRIBUTED FILE SYSTEM

Inventor(s)

Christos Karamanolis
1657 Belleville Way, Apt. J
Sunnyvale, CA 94087

Daniel A. Muntz
10301 Vicksburg Drive
Cupertino, CA 95014

Mallik Mahalingam
620 Park View Drive, #105
Santa Clara, CA 95054

Zheng Zhang
3520 Casabella Court
San Jose, CA 95148

Assignee

Hewlett Packard Company

SEPARATE READ AND WRITE SERVERS IN A DISTRIBUTED FILE SYSTEM

FIELD OF THE INVENTION

The present invention generally relates to distributed file systems, and more particularly to separate servers for reading and writing data in a distributed file system.

BACKGROUND

Distributed file systems are generally employed for storage of large quantities of data and to reduce input/output (I/O) bottlenecks where there are many requests made for file access. In a distributed file system, the file data is spread across multiple data processing systems. File system control and management of file system meta-data is distributed in varying degrees in different systems.

A desirable characteristic of many distributed file systems is scalability. Scalability is a characteristic that refers to the ease with which a distributed file system can be expanded to accommodate increased data access needs or increased storage needs. For example, as additional users are granted access to the distributed file system, new storage servers may be introduced, and the requests of the additional users may be further spread across the old servers and new servers. The scalability of any distributed file system is limited or enhanced by the system design.

Caching is a feature that is commonly used to reduce data access times and to enhance the scalability of distributed file systems. However, caching requires additional management software to address data locking and data consistency issues. Thus, caching introduces additional complexity and overhead into a distributed file system.

Another approach that addresses scalability is the provision of dual paths for access to file data and access to meta-data. In this approach, the meta-data is managed on a server that is separate from the storage servers. However, this approach may create a bottleneck at the meta-data server and thereby restrict scalability.

A system and method that address the aforementioned problems, as well as other related problems, are therefore desirable.

SUMMARY OF THE INVENTION

In various embodiments, a system and method are provided for implementing a distributed file system in which read requests are processed by dedicated read servers and write requests are processed by a dedicated write server. In various embodiments, read requests are separated from write requests and processed by dedicated read servers. A plurality of read servers are coupled to the client applications and each read server reads file data from the distributed file system and returns the file data to the client applications. A write server writes data to the distributed file system. Various embodiments are described for separating read requests from write requests and transmitting read requests to the read servers and write requests to the write server.

It will be appreciated that various other embodiments are set forth in the Detailed Description and Claims which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

Various aspects and advantages of the invention will become apparent upon review of the following detailed description and upon reference to the drawings in which:

FIG. 1 is a functional block diagram that illustrates the flow of file access requests and file data in accordance with one embodiment of the invention;

FIG. 2 is a functional block diagram of a system where separate read and write servers provide access to file data;

FIG. 3 is a functional block diagram of a system where separate read and write servers provide access to file data and a dedicated load balancer distributes the file access requests;

FIG. 4 is a functional block diagram of software components hosted by a read server;

FIG. 5 is a functional block diagram of software components hosted by a write server;

FIG. 6 is a flowchart that illustrates a process performed by the load balancer in processing file access requests;

FIG. 7 is a flowchart of a process performed by a read server in processing requests from a distributed file system client interface;

FIG. 8 is a flowchart of a process performed by the write server in writing data to a file; and

FIGs. 9A, 9B, and 9C, which illustrate successive states of file storage in processing a write request.

DETAILED DESCRIPTION

Various embodiments of the present invention are described in terms of specific functions implemented on specific data processing systems. Those skilled in the art will appreciate, however, that various alternative arrangements of data processing systems and various alternative data processing system architectures could be used to implement the invention.

FIG. 1 is a functional block diagram that illustrates the flow of file access requests and file data in accordance with one embodiment of the invention. System 100 includes a plurality of clients 102-104, a plurality of read servers 106-108, a write server 110, and a distributed file system 112. The clients 102-104 are data processing systems that host client applications (not shown) that issue requests to read data from and write data to storage that is managed by distributed file system 112. Read requests refer to file access operations where file system meta-data, file meta-data, or file data are read from storage, and write requests refer to file access operations where file system meta-data, file meta-data, or file data are written to storage.

To improve scalability and performance, read requests and write requests are processed by different servers. The clients 102-104 send read requests to the read servers 106-108 for processing, and write requests are sent to write server 110. In applications where read activity is much greater than write activity, the separation of read and write servers supports scalability to service more read requests. Measurements from commercial systems indicate that read operations are typically more than 90% of the total operations to a distributed file system. Thus, additional read servers can be coupled to the client applications and to the distributed file system 112 to handle more read requesters. The addition of read servers does not require any reconfiguration of the distributed file system and can be transparent to the user application.

In one embodiment, the particular read server to which a client application sends a read request is selected in a manner that balances the processing load between the read servers. Each read server provides access to all the addressable file storage for each of the coupled client applications.

In an example embodiment, the read servers 106 and 108 and write server 110 are implemented as conventional network file system (NFS) servers that are coupled to

a conventional distributed file system 112 and hosted on separate data processing systems. In another embodiment, the read servers are adapted to receive all file access requests and forward write requests (not shown) to the write server 110. Those skilled in the art will recognize that various alternative remote and distributed file systems could be adapted to operate in accordance with the present invention.

By separating the read and write requests, the system 100 is scalable to process read requests. Since the read servers do not write any data to storage they do not require consistency control of the data, and additional read servers can be added without incurring extra overhead on the other read servers. Since there is only one write server, the overhead associated with maintaining data consistency between multiple writers is eliminated.

FIG. 2 is a functional block diagram of a system 150 where separate read and write servers provide access to file data. Clients systems 102 and 104, read servers 106 and 108, and write server 110 are coupled to network 152. Network 152 is a conventional data communications network through which the servers and clients interact. Storage area network 154 includes the physical storage media on which the file data are stored. Storage Area Networks (SANs) include three types of hardware components: fiber channel switches (e.g., BROCADE FC-SAN Switches), fiber channel adaptors for hosts (e.g., Qlogic FC-SAN host adaptors), and special disks or disk arrays (e.g., Hewlett-Packard's XP512 Disk Arrays). Special software is required to manage and configure SANs, e.g., McData's SAN Management and IBM's Tivoli.

Each of client systems 102 and 104 hosts a client application and an interface to the distributed file system. For example, client 102 hosts client application 156 and distributed file system (DFS) client interface 158. Other than file access requests made by the client application 156, the application-specific functions of the client application are beyond the scope of the present invention.

The DFS client interface 158 is implemented with functional extensions to conventional DFS client software. For example, in one embodiment NFS-client software is extended with functions that separate read requests from write requests and send the requests to the appropriate servers. In another embodiment, the DFS client interface 158 is implemented with conventional NFS-client software, and the read servers 106 and 108 are adapted to separate read requests from write requests. In the latter embodiment, the read servers forward write requests to the write server 110.

Patent/ application number *****, entitled, "EXTENDING A STANDARD-BASED REMOTE FILE ACCESS PROTOCOL AND MAINTAINING COMPATIBILITY WITH A STANDARD PROTOCOL STACK" by Karamanolis et al., filed on January 31, 2001, and assigned to the assignee of the present invention, describes yet another embodiment for implementing the DFS client interface and is hereby incorporated by reference. It will be appreciated that other standards-based or proprietary distributed file systems can be adapted in accordance with the teachings of present invention.

In another embodiment of the invention, the DFS client interface 158 includes functionality that distributes read requests between the read servers 106 and 108 in order to balance the processing load between the read servers. For example, a round-robin or other well known load distribution function can be used to balance read requests between the read servers.

FIG. 3 is a functional block diagram of a system 160 where separate read and write servers provide access to file data and a dedicated load balancer distributes the file access requests. DFS client interface 162 sends read and write requests to load balancer 164 instead of addressing the read and write servers directly. In other respects, DFS interface 162 is implemented as described above.

In one embodiment, load balancer 164 is implemented with a conventional content switch that is coupled to network 152. The load balancer 164 is an application layer switch (i.e., layer 7). Application layer switches that currently support switching for URLs can be programmed or configured to support distributed file system access. The load balancer 164 is configured to receive read and write requests from the DFS client interface 162 components on each of the clients 102-104. In a first embodiment, the load balancer distributes read and write requests to the read servers, and the read servers are configured to forward the write requests to the write server 110. In another embodiment, the load balancer distributes read requests to the read servers and forwards write requests to the write server. Based on the function code present in a file access request, the load balancer distinguishes between read and write requests.

Load balancer 164 attempts to evenly distribute the processing load associated with servicing read requests between the read servers. In one embodiment, a round-robin method is used to distribute the requests to the read servers. More sophisticated approaches may be employed in other embodiments. For example, the load balancer can examine each read request for the quantity of data requested and use the

combination of the quantity of data and number of outstanding read requests to evenly distribute the workload. In another embodiment, each read server reports its workload to the load balancer, and the load balancer uses the relative current workloads of the read servers in distributing read requests.

FIG. 4 is a functional block diagram of software components hosted by a read server 182. Read server 182 is a conventional data processing system having computational and input/output capacity that depend on application requirements. In various embodiments, the software components are conventional or have extensions added to conventional software components.

DFS server 184 receives file access requests from the client application 156. In one embodiment, the DFS server is implemented with conventional server software for a distributed file system, for example, NFS server software. If the DFS client interface 158 or load balancer 164 sends only read requests to the DFS server, the DFS server processes only read requests and commercially available DFS server software is used. In another embodiment, DFS client interface 158 or load balancer 164 sends both read and write requests to the DFS server, and the DFS server is configured to forward write requests to the write server 110.

Physical file system 186 is also implemented with conventional software. For example, the physical file system can be implemented with the Ext2 system of Linux or the FFS of BSD Unix. Alternatively, proprietary software such as NTFS from Microsoft, XFS from Silicon Graphics, or WAFL from Network Appliances, may be used to implement the physical file system..

FIG. 5 is a functional block diagram of software components hosted by a write server. Write server 192 is a conventional data processing system having computational and input/output capacity that depend on application requirements. Comparable to the read server 182, the software components are based on conventional software components.

DFS server 184 processes write requests from the client application 156. The DFS server 184 is adapted to interface with data consistency control element 194. Since the read and write servers have access to the same virtual storage, when file data and meta-data are modified the write server must ensure that the data are modified in a consistent manner. That is, the data and meta-data read by the read servers must be

consistent. "Meta-data" refers to information that describes the file system and information that describes each file. For example, meta-data includes status information, permission levels, physical storage location descriptions, symbolic names, etc.

The data consistency control logic 194 assumes that the client application 156 does not immediately require the most recent data. Once the data consistency control 194 has stored the new meta-data and file data in a consistent state, the new data is accessible to the read servers.

As described below in figures 8 and 9A – 9C, the write server imposes a strict order of operations in accessing the physical storage (e.g., disk) when servicing a write request. This requires support from the physical file system because the physical file system controls data consistency. In one embodiment, the physical file system provides the interface and mechanisms to specify such order requirements. Alternatively, extensions to the physical file system, for example, data consistency control 194, control the order of operations.

FIG. 6 is a flowchart that illustrates a process performed by the load balancer 164 in processing file access requests. At step 302, a file access request is received via network 152 from a DFS client interface 162. In one embodiment, the load balancer is configured to process only read requests (e.g., the DFS client interface separates read requests from write requests), and in another embodiment, the load balancer is configured to process both read and write requests (e.g., the DFS client interface forwards both read and write requests to the load balancer).

In the embodiment where the load balancer receives only read requests, the process continues at step 304 where a read server is selected. As described above, the load balancer attempts to balance the workload between the read servers. For example, the load balancer implements a round-robin or other known load balancing algorithm. At step 306, the request is forwarded to the selected read server, and control returns to step 302 to process the next request.

In the embodiment where the load balancer receives both read and write requests, the process is directed from step 302 to step decision step 308. At decision step 308, the load balancer checks whether the request is a read request or a write request. For read requests, the process is directed to step 304 and the read request is processed as described above. For write requests, the process is directed to step 310

where the write request is sent to the write server 110. The process then returns to step 302 to process the next request.

FIG. 7 is a flowchart of a process performed by a read server in processing requests from a DFS client interface. At step 352, a file access request is received via network 152 from a DFS client interface 158. In one embodiment, the read server is configured to process only read requests (e.g., where the DFS client interface separates read requests from write requests), and in another embodiment, the read server is configured to process read requests and forward write requests to the write server (e.g., where the DFS client interface forwards both read and write requests to the read servers).

In the embodiment where the read server receives only read requests, the process continues at step 354 where a read server is selected as described above. At step 356, the request is forwarded to the selected read server, and control returns to step 302 to process the next request.

In the embodiment where the read server receives both read and write requests, the process is directed from step 352 to step decision step 358. At decision step 358, the read server checks whether the request is a read request or a write request. For read requests, the process is directed to step 354 and the read request is processed as described above. For write requests, the process is directed to step 360 where the write request is sent to the write server 110. The process then returns to step 352 to process the next request.

FIG. 8 is a flowchart of a process performed by the write server in writing data to a file. The process of FIG. 8 is described in conjunction with the block diagrams of FIGs. 9A, 9B, and 9C, which illustrate successive states of file storage in processing a write request. At step 402, a write request is received, either from a read server, a load balancer, or from a DFS client interface, depending on the implementation. The i-node for the file referenced in the write request is read at step 404.

FIG. 9A illustrates the initial state of file storage 500. File storage 500 includes a block bitmap 502, an i-node 504, multiple indirect blocks 506-508, and multiple data blocks 510-512. The i-node includes pointers to the indirect blocks and pointers to a number of data blocks, for example data block 510. The indirect blocks include pointers to data blocks, for example, indirect block 508 references data block 512.

While not shown, it will be appreciated that the file system also includes double and triple indirect blocks as understood by those skilled in the art.

The block bitmap 502 indicates which blocks of file storage 500 have file data and meta-data stored therein, and which blocks are available. The i-node 504 contains information that describes the file, for example, a symbolic name, timestamps, and access permissions.

The information from i-node 504 that is read into memory of the write server at step 404 is shown as block 514. Step 406 conditionally reads blocks of file data if the write request involves updating presently allocated indirect and data blocks.

At step 408, the file data in the write request is used to update data blocks in the memory of the write server. In addition, the i-node and block bitmap are updated in the memory of the write server if necessary (FIG. 9B, 514').

At step 412, the file data from the memory of the write server is written to newly allocated data blocks in file storage 500. For example, in FIG. 9B indirect block 508 and data block 512 are updated and written to file storage 500 as indirect block 508' and data block 512'. In addition, FIG. 9B illustrates a new data block 516 that is written to the file storage. Note that the current i-node 504 still references the old indirect block 508, which references old data block 512. Thus, the read servers continue to have a consistent view of the file data while the write server is processing write requests, even though the data may not be current.

At step 414, the portion of file storage 500 having the block bitmap 502 and i-node 504 are locked, and the updated I-node 504' and block bitmap 502' (FIG. 9C) are written to file storage 500 at step 416. Any old data or indirect blocks are freed at step 416. The block bitmap and i-node areas are unlocked at step 418. FIG. 9C illustrates the state of file storage 500 after the block bitmap and i-node have been updated. The updated i-node 504' references indirect block 508'. Thus, the read servers have access to the new data after the write server completes the i-node update.

At step 420, a response is returned to the DFS client interface, and the process returns to step 402 to receive the next write request.

The present invention is believed to be applicable to a variety of distributed and remote files systems and has been found to be particularly applicable and beneficial with NFS-type file systems. Those skilled in the art will appreciate that the invention is not limited to NFS-type file systems, and other aspects and embodiments of the present invention will be apparent from consideration of the specification and practice of the

invention disclosed herein. It is intended that the specification and illustrated embodiments be considered as examples only, with a true scope and spirit of the invention being indicated by the following claims.